Projet PageRank

Équipe EF7 : Evann DREUMONT et Timothée KLEIN

Décembre 2023

Table des matières

1 Introduction

Ce fichier vise à expliquer le fonctionnement et les choix que nous avons réalisés dans la réalisation du programme de PageRank visant a noter les pages webs en fonction de leur référencement.

2 Liste des modules

Deux sous-modules sont implémentés :

- Module Matrice permettant la gestion de matrices de taille quelconque
- Module Trifusion permettant de réaliser le trifusion

3 Raffinages

3.1 Page Rank

3.1.1 Description

L'algorithme PageRank permet de calculer la popularité de pages webs en fonction de leurs référencements, stockés dans un graphe. Il fournit en sortie deux fichiers .prw et .pr, qui sont respectivement les poids des pages et les pages ordonées selon leur poids (plus haut poids -> meilleure popularité).

3.1.2 Raffinages

R0 : Calculer le Rang de chaque page ainsi que le poids de chacune des pages dans un graphe avec des matrices pleines

Fichier Graphe	Fichier PageRank	Fichier Poids
6 0 1 0 2 2 0 2 1 2 4 3 4 3 5 4 3 4 5 5 3	3 5 4 1 2 0	6 0.85000000000000 150 0.34870368521482 0.26859608185466 0.19990381197332 0.07367926270376 0.05741241249643 0.05170474575702

Table 1 – Exemple de l'algorithme PageRank

R1 : **Comment** "Calculer le Rang de chaque page ainsi que le poids de chacune des pages dans un graphe avec des matrices pleines ?"

Gérer les arguments avec lesquels le programme est lancé
out : *Entier* alpha ; *Flottants* k, eps ; *Booléens* plein, valide ; *String* prefixe ; *String* reseau
Si valide **Alors**

Charger le graphe dans une matrice d'adjacence pondérée
in : *String* reseau ; **out** : *Matrice* H, *Tableau* sortants, *Entier* n
Appliquer l'algorithme PageRank jusqu'à terminaison **in** : H, n, alpha, k, eps ; **out** : *Matrice* Pi
Sauvegarder les résultats **in** : Pi, n, alpha, k, prefixe

Fin Si

R2 : **Comment** "Gérer les arguments avec lesquels le programme est lancé" ?

valide <- Vrai
Initialiser les valeurs par défaut aux arguments **out** : alpha, k, eps, prefixe, *Booléen* plein
Récupérer les arguments fournis **out** : alpha, k, eps, plein, prefixe, reseau.
Vérifier si les arguments sont valides **in** : alpha, k, eps ; **out** : *Booléen* valide

R2 : **Comment** "Charger le graphe dans une matrice d'adjacence pondérée" ?

Ouvrir reseau
n <- *Entier* sur la première ligne du fichier
H <- *Matrice* n x n vide
Stocker dans H chaque référencement **in** : reseau, n ; **out** : H, *Tableau* sortants
Fermer reseau
Pondérer les lignes **in** : n, sortants ; **in-out** : H

R2 : **Comment** "Appliquer l'algorithme PageRank jusqu'à terminaison" ?

Créer la matrice S **in** n ; **out** : *Matrice* S
Créer la matrice G **in** : S, n, alpha ; **out** : *Matrice* G
Calculer la matrice Pi par itérations **in** : G, k, eps **out** : Pi

R2 : **Comment** "Sauvegarder les résultats" ?

Trier le vecteur Pi **in-out** : *Matrice* Pi
Ecrire n, alpha et k à la première ligne dans le fichier poids (prefixe.prw)
Pour chaque ligne du vecteur Pi

Ecrire dans une nouvelle ligne du fichier poids (prefixe.prw) la valeur de la ligne
Ecrire dans une nouvelle ligne du fichier PageRank (prefixe.pr) l'index de la ligne

Fin Pour

R3 : **Comment** "Initialiser les valeurs par défaut aux arguments" ?

alpha <- 0.85
k <- 150
plein <- Vrai
eps <- 0.0
prefixe <- "output"

R3 : **Comment** "Récupérer les arguments fournis" ?

i<-0
Tant Que i < (Nombre d'arguments-1) **faire**

Selon Argument(i) **Dans**

"-P" -> plein <- Vrai ; i<-i+1
"-C" -> plein <- Faux ; i<-i+1
"-A" -> alpha <- Argument(i+1) ; i<-i+2

"-E" -> eps <- Argument(i+1) ; i<-i+2
"-K" -> k <- Argument(i+1) ; i<-i+2
"-R" -> prefixe <- Argument(i+1) ; i<-i+2

Fin Selon

Fin Tant Que
reseau <- Argument(i)

R3 : **Comment** "Vérifier si les arguments sont valides" ?

Si alpha>1 **Ou Alors** alpha<0 **Ou Alors** k<0 **Ou Alors** eps<0 **Ou Alors** valide = Faux **Ou Alors** reseau n'est pas accessible **Alors**

Ecrire("Un argument a une valeur illégale, veuillez relire votre appel")
valide <- Faux

Fin Si

R3 : **Comment** "Stocker dans H chaque référencement" ?

sortants <- *Tableau* vide de taille n
Pour chaque ligne de reseau **Faire**

i <- premier nombre de la ligne
j <- deuxieme nombre de la ligne
H(i,j) <- H(i,j) + 1
sortants[i] <- sortants[i] + 1

Fin Pour

R3 : **Comment** "Pondérer les lignes" ?

Pour i de 1 à n pas 1 **faire**

Pour j de 1 à n pas 1 **faire**

H(i,j)<- H(i,j)/sortants[i]

Fin Pour

Fin Pour

R3 : **Comment** "Créer la matrice S" ?

S <- copie(H)
Pour i de 1 à n pas 1 **Faire**

Si sortants[i]=0 **Alors**

Pour j de 1 à n pas 1 **Faire**

S(i,j) <- 1/n

Fin Pour

Fin Si

Fin Pour

R3 : **Comment** "Créer la matrice G" ?

Attila <- *Matrice* n x n remplie de huns
G <- alpha * S + Attila * (1-alpha)/n

R3 : **Comment** "Calculer la matrice Pi par itérations"?

```
i <- 1
Pi_avant <- Matrice n x 1 remplie de 1/n
Pi <- Pi_avant * G
Tant Que i<k Et || Pi - Pi_avant || > eps Faire

        Pi_avant <- Pi
        Pi <- Pi * G

Fin TQ
```

3.2 Module Matrice

3.2.1 Description

Ce module fournit le type Matrice et permet des opérations élémentaires sur celui-ci. De plus, le module est générique et sera implémenté de telle sorte qu'il prend en paramètre le type de structure utilisé, c'est-à-dire si c'est une matrice pleine ou une matrice creuse.

3.2.2 Raffinages

R0 : Multiplier deux matrices carrées A et B de taille n,p et p,l

R1 : **Comment** "Multiplier deux matrices carrées A et B de taille n,p et p,l"?

Initialiser une matrice M vide de taille n,l	**out** : *Matrice* M
Remplir cette matrice avec les coefficients du produit	**in-out** : *Matrice* M
Renvoyer M	**in** : *Matrice* M

R2 : **Comment** "Remplir cette matrice avec les coefficients du produits"?

```
Pour i allant de 1 à n Faire

    Pour j allant de 1 à l faire

        M(i,j) <- Somme pour k allant de 1 à p de A(i,k)*B(k,j)

    Fin Pour

Fin Pour
```

R0 : Ajouter deux matrices A et B de taille n,p

R1 : **Comment** "Ajouter deux matrices A et B de taille n,p"?

Initialiser une matrice M vide de taille n,p	**out** : *Matrice* M
Remplir cette matrice avec les coefficients de la somme	**in-out** : *Matrice* M
Renvoyer M	**in** : *Matrice* M

R2 : **Comment** "Remplir cette matrice avec les coefficients de la somme"?

```
Pour i allant de 1 à n faire

    Pour j allant de 1 à p faire

        M(i,j) <- A(i,j)+B(i,j)

    Fin Pour
```

Fin Pour

R0 : Multiplier une matrice A par un scalaire

R1 : **Comment** "Multiplier une matrice A par un scalaire k" ?

Pour i allant de 1 à n **faire**

Pour j allant de 1 à p **faire**

A(i,j) <- A(i,j)*k

Fin Pour

Fin Pour

R0 : Transposer une matrice A de taille n,p

R1 : **Comment** "Transposer une matrice A de taille n,p" ?

Initialiser une matrice M vide de taille p,n — **out** : *Matrice* M
Remplir M — **in-out** : *Matrice* M
Renvoyer M — **in** : *Matrice* M

R2 : **Comment** "Remplir M" ?

Pour i allant de 1 à p **faire**

Pour j allant de 1 à n **faire**

M(i,j) <- A(j,i)

Fin Pour

Fin Pour

3.3 Module Trifusion

3.3.1 Description

Module permettant de trier un vecteur selon un ordre décroissant avec la méthode du tri fusion.

3.3.2 Raffinages

R0 : Trier un vecteur selon un ordre décroissant avec un opérateur d'ordre.

R1 : **Comment** "Trier un vecteur selon un ordre décroissant avec un opérateur d'ordre."

début <- 1
fin <- Taille du vecteur
Appeler l'algorithme récursif de tri — **in-out** : *Matrice* : vecteur, **in** : *Entiers* : début, fin ; *Fonction* Opérateur

R2 : **Comment** "Appeler l'algorithme récursif de tri"

Si non début > fin **Alors**

milieu <- (début + fin) / 2
Appeler l'algorithme récursif de tri avec vecteur, début et milieu
Appeler l'algorithme récursif de tri avec vecteur, milieu+1 et fin
Fusion des deux moitiés triées — **in-out** :Vecteur ; **in** : début, milieu, fin, Opérateur

Fin Si

R3 : **Comment** "Fusionner les deux moitiés triées"

vecteur_trie <- Copie(vecteur)
i <- début
j <- milieu + 1
Tant Que i < milieu+1 **Et** j < fin+1 **faire**

Copier min(vecteur[i],vecteur[j]) dans vecteur_trie et incrémenter i ou j en fonction
in-out : i,j ; **in** : Opérateur, Vecteur ; **out** : vecteur_trie

Fin TQ
Recopier vecteur_trie dans vecteur **in-out** : vecteur_trie ; **in** : i,j ;**out** : Vecteur

R4 : **Comment** "Copier min(vecteur[i],vecteur[j]) et incrémenter i ou j" ?

Si vecteur[i] et vecteur[j] < vecteur[j] **Alors**

vecteur_trie[j-milieu+i]<- vecteur[i]
i <- i+1

Sinon

vecteur_trie[j-milieu+i]<- vecteur[j]
j <- j+1

Fin Si

R4 : **Comment** "Recopier vecteur_trie dans vecteur" ?

Si i=milieu +1 Alors

vecteur[deb :j]<-vecteur_trie[deb :j]

Sinon

vecteur_trie[fin-(milieu-i) :fin+1] <- vecteur[i :milieu+1]
vecteur[début :fin] <- vecteur_trie[début :fin]

Fin Si